Partial Versus Total Halting in P Systems

Artiom Alhazov[1], Rudolf Freund[2], Marion Oswald[2], Sergey Verlan[3]

[1] Department of Information Technologies
Abo Akademi University
Turku Center for Computer Science
FIN-20520 Turku, Finland
aalhazov@abo.fi
and
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028, Moldova
aartiom@math.md

[2] Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
{rudi,marion}@emcc.at

[3] LACL, Département Informatique
UFR Sciences et Technologie
Université Paris XII
61, av. Général de Gaulle
94010 Créteil, France
verlan@univ-paris12.fr

Summary. We consider a new variant of the halting condition in P systems, i.e., a computation in a P system is already called halting if not for all membranes a rule is applicable anymore at the same time, whereas usually a computation is called halting if no rule is applicable anymore in the whole system. This new variant of partial halting is especially investigated for several variants of P systems working in different derivation modes.

1 Introduction

In the seeding papers of Gheorghe Păun (e.g., see [19], [9]) introducing membrane computing, membrane systems were introduced as systems with a hierarchical (tree-like) structure and the rules being applied in a maximally parallel manner; the results were taken as the contents of a specified output membrane in the final configurations of halting computations, i.e., at the end of computations to which no rule was applicable anymore. In this paper, we investigate a new variant of halting – *partial halting* –, see [13], i.e., we consider a computation to halt as soon

as not for all membranes a rule is applicable anymore at the same time. Moreover, we especially also consider the derivation mode of minimal parallelism (e.g., see [7]), i.e., for each membrane, at least one rule – if possible – has to be applied, but it is not required to use a maximal multiset of rules. Finally, in the asynchronous derivation mode an arbitrary number of rules can be applied in parallel, and in the sequential derivation mode exactly one rule has to be applied in each computation step.

The paper is organized as follows: We first recall some well-known definitions, notions, and results for matrix grammars and register machines and then define a special model of P systems – *P systems with permitting contexts* – that covers a lot of variants known from the literature such as antiport P systems, P systems with conditional uniport rules, evolution/communication P systems, and P systems with boundary rules. For establishing our results, we especially consider the new stopping mode of partial halting; we first state some general result and then show that P systems using membrane rules with permitting contexts working in the minimally parallel mode and with partial halting can only generate matrix languages. On the other hand, we improve or newly establish results showing that specific variants of P systems with permitting contexts such as antiport P systems, P systems with conditional uniport rules, and evolution/communication P systems together with the newly introduced variant of minimal parallelism and with total halting are computationally complete.

2 Definitions

In this section, we first recall some basic notions and notations and then give precise definitions for matrix grammars, register machines, and a general model of P systems using membrane rules with permitting contexts as they are considered in this paper; moreover, we show how several well-known models of P systems (P systems with symport/antiport rules, P systems with conditional uniport rules, evolution/communication P systems, P systems with boundary rules) can be interpreted as special variants of this general model.

2.1 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to [8] and [23]. We just list a few notions and notations: $\mathbb{N}$ denotes the set of non-negative integers. V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; by NRE and $NREG$ we denote the family of recursively enumerable sets and regular sets of non-negative integers, respectively.

Let $\{a_1, ..., a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to $(a_1, ..., a_n)$ is $\left(|x|_{a_1}, ..., |x|_{a_n}\right)$. The *Parikh image* of a language L over $(a_1, ..., a_n)$

is the set of all Parikh vectors of strings in L. For a family of languages F, the family of Parikh images of languages in F is denoted by PsF. A (finite) multiset $\langle m_1, a_1\rangle \ldots \langle m_n, a_n\rangle$ with $m_i \in \mathbb{N}$, $1 \leq i \leq n$, can be represented by any string x the Parikh vector of which with respect to $(a_1, ..., a_n)$ is $(m_1, ..., m_n)$; if $m_i = 1$ for all $1 \leq i \leq n$, then x can also be represented by the corresponding set $\{m_1, ..., m_n\}$.

The family of recursively enumerable languages is denoted by RE, the family of context-free and regular languages by CF and REG, respectively. The corresponding families of languages over a k-letter alphabet are denoted by $X(k)$, $X \in \{RE, CF, REG\}$; for $k = 1$ we obtain $PsX(1) = NX$ and, moreover, $NREG = NCF$.

2.2 Matrix Grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over (N, T). For $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is

$$L(G) = \{w \in T^* \mid S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w,\ w_j \in (N \cup T)^*,\ m_{i_j} \in M \text{ for } 1 \leq j \leq k, k \geq 1\}.$$

The family of languages generated by matrix grammars without appearance checking (over a one-letter alphabet) is denoted by MAT^{λ} ($MAT^{\lambda}(1)$). It is known that $CF \subset MAT^{\lambda} \subset RE$ as well as $PsCF \subset PsMAT^{\lambda} \subset PsRE$, and especially $NREG = NCF = PsMAT^{\lambda}(1) \subset NRE$. For further details about matrix grammars we refer to [8] and to [23].

2.3 Register Machines

The proofs of the results establishing computational completeness in the area of P systems often are based on the simulation of register machines; we refer to [17] for original definitions, and to [11] for definitions like those we use in this paper:

An *n-register machine* is a construct $M = (n, B, P, p_0, p_h)$, where n is the number of registers, B is a set of labels for injectively labelling the instructions in P, p_0 is the initial/start label, and p_h is the final label.

The instructions are of the following forms:

– $p : (A(r), q, s)$ (ADD instruction)
  Add 1 to the contents of register r and proceed to one of the instructions (labelled with) q and s.

- $p : (S(r), q, s)$ (SUB instruction)
  If register r is not empty, then subtract 1 from its contents and go to instruction q, otherwise proceed to instruction s.
- $p_h : halt$ (HALT instruction)
  Stop the machine. The final label l_h is only assigned to this instruction.

A (non-deterministic) register machine M is said to generate a vector $(n_1, ..., n_\beta)$ of natural numbers if, starting with the instruction with label p_0 and all registers containing the number 0, the machine stops (it reaches the instruction $p_h : halt$) with the first β registers containing the numbers $n_1, ..., n_\beta$ (and all other registers being empty).

Without loss of generality, in the succeeding proofs we will assume that for non-deterministic register machines in each ADD instruction $p : (A(r), q, s) \in P$ and in each SUB instruction $p : (S(r), q, s) \in P$ the labels p, q, s are mutually distinct (for a proof see [16]).

A register machine is called *deterministic* if and only if in every ADD instruction $p : (A(r), q, s) \in P$ we have $q = s$; in this case we also write $p : (A(r), q)$ instead. A deterministic register machine M is said to accept a vector $(n_1, ..., n_\beta)$ of natural numbers if, starting with the instruction with label p_0 and registers 1 to β containing the numbers $n_1, ..., n_\beta$, the machine stops (it reaches the instruction $p_h : halt$) with the all registers being empty.

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of non-negative integers which can be generated by Turing machines, i.e., the family $PsRE$.

The results proved in [10] (based on the results established in [17]) and [11], [14] immediately lead to the following results:

Proposition 1. *For any recursively enumerable set $L \subseteq \mathbb{N}^\beta$ of vectors of non-negative integers there exists a non-deterministic $(\beta + 2)$-register machine M generating L in such a way that, when starting with all registers 1 to $\beta+2$ being empty, M non-deterministically computes and halts with n_i in registers i, $1 \leq i \leq \beta$, and registers $\beta + 1$ and $\beta + 2$ being empty if and only if $(n_1, ..., n_\beta) \in L$. Moreover, the registers 1 to β are never decremented.*

Proposition 2. *For any recursively enumerable set $L \subseteq \mathbb{N}^\beta$ of vectors of non-negative integers there exists a deterministic $(\beta + 2)$-register machine M accepting L in such a way that, when starting with $n_1, ..., n_\beta$ in registers 1 to β and with register $\beta + 1$ and $\beta + 2$ being empty, M halts with all registers being empty if and only if $(n_1, ..., n_\beta) \in L$.*

2.4 A General Model of P Systems with Permitting Contexts

We now introduce a general model of P systems with permitting contexts covering the most important models of communication P systems as well as evolution/communication P systems. For the state of the art in the P systems area, we refer to the P systems web page [25].

A *P system (of degree* d, $d \geq 1$*) with permitting contexts* (in the following also called P system for short) is a construct

$$\Pi = (V, T, E, \mu, w_0, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o) \text{ where}$$

1. V is an alphabet; its elements are called *objects*;
2. $T \subseteq V$ is an alphabet of *terminal objects*;
3. $E \subseteq V$ is the set of objects occurring in an unbounded number in the environment;
4. μ is a *membrane structure* consisting of d membranes (usually labelled with i and represented by corresponding brackets $[_i$ and $]_i$, $1 \leq i \leq d$);
5. w_i, $1 \leq i \leq d$, are strings over V associated with the regions $1, 2, \ldots, d$ of μ; they represent multisets of objects initially present in the regions of μ; w_0 represents the multiset of objects from $V \setminus E$ initially present in the environment (in the following we usually shall assume $w_0 = \lambda$);
6. R_i, $1 \leq i \leq d$, are finite sets of *membrane rules with permitting contexts* over V associated with the membranes $1, 2, \ldots, d$ of μ; these evolution rules in R_i are of the form ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z}$, where $w, z \in V^*$ are the contexts in the region outside membrane i and inside membrane i, respectively, u outside membrane i is replaced by v and x inside membrane i is replaced by y;
7. i_o is a number between 1 and d and it specifies the *output* membrane of Π.

The rule ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z}$ from R_i is applicable if and only if the multiset uw occurs in the region outside membrane i (in the following also denoted by $\hat{\imath}$) and the multiset xz occurs in the region inside membrane i. The application of this rule results in subtracting the multiset identified by u from the multiset in $\hat{\imath}$ and adding v instead as well as subtracting x and adding y in the region inside membrane i. The permitting contexts w and z themselves or subsets of w and z can be (part of) permitting contexts in other rules and, moreover, even be modified by another rule in the same derivation step. On the other hand, any object can be modified, i.e., be part of u or x in a rule ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z}$, by only one application of one rule in each derivation step. The rules to be applied in parallel and the objects to be modified by these rules are chosen in a non-deterministic way.

Instead of writing ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z} \in R_i$ we can also write ${}^{u}_{w}[_i {}^{x}_{z} \rightarrow {}^{v}_{w}[_i {}^{y}_{z}$ and in this way collect all rules from the R_i, $1 \leq i \leq d$, in one single set of rules $R = \left\{ {}^{u}_{w}[_i {}^{x}_{z} \rightarrow {}^{v}_{w}[_i {}^{y}_{z} \mid {}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z} \in R_i \right\}$.

The membrane structure and the multisets represented by w_i, $0 \leq i \leq d$, in Π constitute the *initial configuration* of the system.

In the *maximally parallel derivation mode*, a transition from one configuration to another one is obtained by the application of a maximal multiset of rules, i.e., no additional rules could be applied anymore to the objects occurring in the current configuration. The system continues maximally parallel derivation steps until there remain no applicable rules in any region of Π; then the system halts (*total halting*). We consider the number of objects from T contained in the output membrane i_o at the moment when the system halts as the *result* of the underlying

computation of Π yielding a vector of non-negative integers for the numbers of terminal symbols in the output membrane i_0; observe that here we do not count the non-terminal objects present in the output membrane. The set of results of all halting computations possible in Π is denoted by $Ps(\Pi)$, respectively. Below, we shall consider variants of P systems using only rules of very restricted types α. The family of all sets of vectors of non-negative integers computable by P systems with d membranes and using rules of type α is denoted by $Ps_gOP_d(\alpha, max, H)$.

When using the *minimally parallel derivation mode*, in each derivation step we choose a multiset of rules from the R_i in such way that to this chosen multiset no rule from a set R_j from which no rule has been taken so far, could be added anymore to be applied in parallel with the rules already chosen.

In the *asynchronous* and the *sequential derivation mode*, in each derivation step we apply an arbitrary number of rules/ exactly one rule, respectively. The corresponding families of sets of vectors of non-negative integers generated by P systems with d membranes and using rules of type α are denoted by $Ps_gOP_d(\alpha, X, H)$, $X \in \{min, asyn, sequ\}$.

If instead of the total halting we take *partial halting*, i.e., computations halting as soon as in at least from one set of rules no rule is applicable anymore, the corresponding families are denoted by $Ps_gOP_d(\alpha, X, h)$, $X \in \{max, min, asyn, sequ\}$.

All these variants of P systems can also be considered as accepting devices, the input being given as the numbers of objects in the distinguished membrane i_0. The corresponding families of sets of vectors of non-negative integers accepted by P systems with d membranes and using rules of type α are denoted by $Ps_aOP_d(\alpha, X, Y)$, $X \in \{max, min, asyn, sequ\}$, $Y \in \{H, h\}$. In this case, it also makes sense to consider deterministic P systems, i.e., systems where for each configuration obtained in this system we can derive at most one configuration. The corresponding families are denoted by $DPs_aOP_d(\alpha, X, Y)$.

If we only count the number of terminal objects and do not distinguish between different (terminal) objects, in all the definitions given above, we replace Ps by N. When the parameter d is not bounded, it is replaced by $*$.

In the following, we now consider several restricted variants of membrane rules with permitting contexts well known from the literature.

P systems with symport/antiport rules

For definitions and results concerning P systems with symport/antiport rules, we refer to the original paper [18] as well as to the overview given in [22]. An *antiport rule* is a rule of the form ${}^{u}[^{x} \rightarrow {}^{x}[^{u}$ usually written as $(x, out; u, in)$, $ux \neq \lambda$. A *symport rule* is of the form $[^{x} \rightarrow {}^{x}[$ or ${}^{u}[\rightarrow [^{u}$ usually written as (x, out), $x \neq \lambda$, or (u, in), $u \neq \lambda$, respectively.

The weight of the antiport rule $(x, out; u, in)$ is defined as $\max\{|x|, |u|\}$. Using only antiport rules with weight k induces the type α usually written as $anti_k$. The weight of a symport rule (x, out) or (u, in) is defined as $|x|$ or $|u|$, respectively. Using only symport rules with weight k induces the type α usually written as

sym_k. If only antiport rules $(x, out; u, in)$ of weight ≤ 2 and with $|x| + |u| \leq 3$ as well as symport rules of weight 1 are used, we shall write $anti_{2'}$.

P systems with conditional uniport rules

A *conditional uniport rule* is a rule of one of the forms ${}^{ab}[\to {}^{b}[^{a}$, $[^{ab} \to {}^{a}[^{b}$, ${}^{a}[^{b} \to [^{ab}$, ${}^{b}[^{a} \to {}^{ab}[$, with $a, b \in V$; in every case, the object a is moved across the membrane, whereas the object b stays where it is. Using only rules of that kind induces the type $uni_{1,1}$. Conditional uniport rules were first considered in [24] for the case of tissue P systems, showing computational completeness with maximal parallelism and total halting (using 24 cells).

P systems with boundary rules and evolution/communication P systems

In P systems with boundary rules as defined in [4], evolution rules as well as communication rules with permitting contexts are considered. Usually, we only consider evolution rules that are non-cooperative, i.e., of the form $a \to v$ with $a \in V$ and $v \in V^*$; a rule $a \to v \in R_i$ corresponds to $[^{a} \to [^{v} \in R_i$ in our general notation. The communication rules are symport or antiport rules with permitting contexts, i.e., of the form ${}^{u}_{w}[^{x}_{z} \to {}^{x}_{w}[^{u}_{z}$.

In [5], boundary rules of the form ${}^{u}[^{x} \to {}^{v}[^{y}$ are considered, i.e., rewriting on both sides of the membrane. In evolution/communication P systems as introduced in [6], we allow non-cooperative evolution rules as well as antiport (of weight k) and symport rules (of weight l), and we denote this type of rules by $(ncoo, anti_k, sym_l)$.

3 Results

After recalling some general results for the new variant of *partial halting* already established in [13], which immediately yield comparable computational completeness results in the case of antiport P systems for total and partial halting, we prove that P systems with permitting contexts working in the sequential, in the asynchronous or even in the minimally parallel derivation mode and with partial halting can only generate Parikh sets of matrix languages (regular sets of non-negative integers). On the other hand, specific variants of P systems with permitting context such as P systems with antiport rules, P systems with symport rules, P systems with conditional uniport rules, and evolution/communication P systems together with the newly introduced variant of minimal parallelism and with total halting are computationally complete.

3.1 General Observations

Looking carefully into the definitions of the derivation modes as well as the halting modes explained above, we observe the following general results already established in [13]:

Theorem 1. *Any variant of P systems yielding a family of sets of non-negative integers F when working in the derivation mode X, $X \in \{max, min, asyn, sequ\}$, with only one set of rules assigned to a single membrane and stopping with total halting yields the same family F when working in the derivation mode X with only one set of rules assigned to a single membrane when stopping with partial halting, too.*

Theorem 2. *Any variant of P systems yielding a family of sets of non-negative integers F when working in the derivation mode X, $X \in \{asyn, sequ\}$, with only one set of rules assigned to a single membrane and stopping with total or partial halting, respectively, yields the same family F when working in the minimally parallel derivation mode and stopping with the corresponding halting mode, too.*

For any P system using rules of type α, with a derivation mode X, $X \in \{min, asyn, sequ\}$, and partial halting, we only get Parikh sets of matrix languages (regular sets of non-negative integers):

Theorem 3. *For every $X \in \{min, asyn, sequ\}$,*
$Ps_gOP_*(\alpha, X, h) \subseteq PsMAT^\lambda$ and $N_gOP_*(\alpha, X, h) \subseteq NREG$.

Proof. We only prove $Ps_gOP_*(\alpha, X, h) \subseteq PsMAT^\lambda$; the second inequality $N_gOP_*(\alpha, X, h) \subseteq NREG$ is a direct consequence of the first one, having in mind that $NREG = PsMAT^\lambda(1)$. Hence, let us start with a P system $\Pi = (V, T, E, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o)$ using rules of a specific type α, working with the derivation mode X. The stopping condition h – partial halting – then guarantees that in order to continue a derivation there must exist a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $1 \leq i \leq d$, such that all these rules are applicable in parallel. We now consider all functions δ with $\delta(i, r) \in \{0, 1\}$ and $\delta(i, r) = 1$ if and only if the rule $r \in R_i$, $1 \leq i \leq d$, is assumed to be applicable to the current sentential form in a matrix grammar $G_M = \left(V_M, \overline{T}, S, M\right)$ generating representations of all possible configurations computable in the given P system Π with the representation of an object a in membrane i as (i, a). We start with the matrix $(S \to Kh(w))$ where $h(w)$ is a representation of the initial configuration. A derivation step in Π then is simulated in G_M as follows:

(i) We non-deterministically choose some δ as described above and use the matrix $(K \to K(\delta))$. Afterwards, we use the matrix $(K(\delta) \to K'(\delta), s_1, ..., s_m)$ where each subsequence s_j, $1 \leq j \leq m$, checks the applicability of a rule $r \in R_i$ with $\delta(i, r) = 1$. For checking the applicability of $\begin{smallmatrix} u \\ w \end{smallmatrix}\left[\begin{smallmatrix} x \\ z \end{smallmatrix} \to \begin{smallmatrix} v \\ w \end{smallmatrix}\right[\begin{smallmatrix} y \\ z \end{smallmatrix} \in R_i$, we have to check for the appearance of uw in membrane $\hat{i}$ (the outer region of membrane i) and for the appearance of xz in the (inner) region of membrane i. This can be done by the subsequence $((\hat{i}, uw) \to (\hat{i}, \overline{uw}), (\hat{i}, \overline{uw}) \to (\hat{i}, uw), (i, xz) \to (i, \overline{xz}), (i, \overline{xz}) \to (i, xz))$, where $(i, v) \to (i, \overline{v})$, for $v = v_1 ... v_h$, $v_j \in V$, $1 \leq j \leq h$, $h \geq 0$, is a shortcut for the sequence $((i, v_1) \to (i, \overline{v_1}), ..., (i, v_h) \to (i, \overline{v_h}))$ etc.

(ii) After that, we non-deterministically guess a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $r_i = \begin{smallmatrix} u(i) \\ w(i) \end{smallmatrix}\left[\begin{smallmatrix} x(i) \\ z(i) \end{smallmatrix} \to \begin{smallmatrix} v(i) \\ w(i) \end{smallmatrix}\right[\begin{smallmatrix} y(i) \\ z(i) \end{smallmatrix}$, and $\delta(i, r_i) = 1$, $1 \leq$

$i \leq d$, such that all these rules are applicable in parallel. This can be checked by the corresponding matrix $(K'(\delta) \to K''(\delta), t_1, ..., t_d, t'_1, ..., t'_d)$ with the subsequences t_i, t'_i, $1 \leq i \leq d$, being defined (in the shortcut notation as above) by $t_i = \left((\hat{\imath}, u(i)) \to \left(\hat{\imath}, \overline{u(i)}\right), (i, x(i)) \to \left(i, \overline{x(i)}\right)\right)$ and $t'_1 = \left(\left(\hat{\imath}, \overline{u(i)}\right) \to (\hat{\imath}, u(i)), \left(i, \overline{x(i)}\right) \to (i, x(i))\right)$. Observe that only the objects in $u(i)$ and $x(i)$ are assigned to the rule r_i, whereas the permitting contexts $w(i)$ and $z(i)$ may be contexts for another rule or be affected themselves by another rule, and, moreover, that the applicability of the rules themselves has already been checked in (i).

(iii) Finally, we take different matrices depending on the derivation mode:

1. In the sequential derivation mode, we only have to take all possible matrices simulating the application of one rule ${}^{u}_{w}[^{x}_{z} \to {}^{v}_{w}[^{y}_{z} \in R_i$ with $\delta(i, r) = 1$: $(K''(\delta) \to Kh_{\hat{\imath}}(v) h_i(y), (\hat{\imath}, u) \to \lambda, (i, x) \to \lambda)$, where the morphisms h_j are defined by $h_j(a) = (j, a)$, $0 \leq j \leq d$, $a \in V$, except $h_0(a) = \lambda$ for $a \in E$ (these symbols, by definition, are available in an unbounded number in the environment).
2. In the asynchronous derivation mode, we have to allow an arbitrary number of rules to be applied in parallel; we simulate the application of rules sequentially, priming the results such that they cannot be used immediately. Finally, if for the current derivation step, the application of no further rule is intended, we can deprime the result symbols to be available for the simulation of the next derivation step. In sum, we use the matrices $(K''(\delta) \to K'''(\delta))$, $(K'''(\delta) \to K'''(\delta) h'_{\hat{\imath}}(v) h'_i(y), (\hat{\imath}, u) \to \lambda, (i, x) \to \lambda)$ – where the morphisms h'_j are defined by $h'_j(a) = (j, a')$, $0 \leq j \leq d$, $a \in V$, except $h'_0(a) = \lambda$ for $a \in E$ – for every rule ${}^{u}_{w}[^{x}_{z} \to {}^{v}_{w}[^{y}_{z} \in R_i$ with $\delta(i, r) = 1$, as well as $\left(K'''(\delta) \to \overline{K}(\delta)\right)$, $\left(\overline{K}(\delta) \to \overline{K}(\delta), (j, a') \to (j, a)\right)$, $0 \leq j \leq d$, $a \in V$, and finally $\left(\overline{K}(\delta) \to K\right)$.
3. For the minimally parallel mode, instead of $(K''(\delta) \to K'''(\delta))$ as in 2, we simulate the application of a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $1 \leq i \leq d$, $r_i = {}^{u(i)}_{w(i)}[^{x(i)}_{z(i)} \to {}^{v(i)}_{w(i)}[^{y(i)}_{z(i)}$, and $\delta(i, r_i) = 1$ such that all these rules are applicable in parallel, which is accomplished by the matrix $\Big(K''(\delta) \to K'''(\delta) h'_{\hat{1}}(v) h'_1(y) ... h'_{\hat{d}}(v) h'_d(y), \left(\hat{1}, u(1)\right) \to \lambda, (1, x(1)) \to \lambda, ..., \left(\hat{d}, u(d)\right) \to \lambda, (d, x(d)) \to \lambda\Big)$.

As a technical detail we have to mention that it does not matter whether all the primed symbols are deprimed again, this would just make them unavailable during the next steps. Any sentential form containing primed symbols is considered to be non-terminal, hence, it cannot contribute to $L(G_M)$. Moreover, we have to point out that every symbol $e \in E$ from the environment being available there in an unbounded number neither needs to be checked for appearance in $\hat{1}$ ($= 0$) nor to be generated/eliminated or primed/deprimed, i.e., rules like $(0, e) \to \lambda$, $(0, e) \to (0, \overline{e})$, $(0, \overline{e}) \to (0, e)$ have to be omitted.

Finally, we may stop the simulation of computation steps of Π and use the matrices $(K \to F)$, $(F \to F, (i,a) \to (i,\overline{a}))$ for every object a and every membrane i, and the final matrix $(F \to \lambda)$ for generating a terminal string of G_M.

Now, we have to extract the representations of final configurations from $L(G_M)$: For every possibility of choosing a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $1 \leq i \leq d$, such that all these rules are applicable in parallel, we construct a regular set checking for the applicability of this sequence in any possible representation of configurations of Π; then we take the union of all these regular sets and take its complement thus obtaining a regular set R. In $L(G_M) \cap R$ we then find at least one representation for every final configuration of computations in Π, but no representation of a non-final configuration.

Finally, let g be a projection with $g((i,\overline{a})) = \lambda$ for every $i \neq i_0$ as well as $g((i_0,\overline{a})) = \lambda$ for $a \in V \setminus T$ and $g((i_0,\overline{a})) = a$ for $a \in T$. Due to the closure properties of MAT^{λ}, we obtain $Ps(g(L(G_M) \cap R)) = Ps(\Pi) \in PsMAT^{\lambda}$.

3.2 Results for Symport/Antiport Systems

The following results are well known (e.g., see [20]; for an overview of actual results also see [22]):

Theorem 4. $Ps_gOP_1(anti_{2'}, max, H) = DPs_aOP_1(anti_{2'}, max, H) = PsRE$.

Theorem 5. *For every* $X \in \{asyn, sequ\}$,
$Ps_gOP_*(anti_*, X, H) = Ps_gOP_1(anti_{2'}, X, H) = PsMAT^{\lambda}$ *and*
$N_gOP_*(anti_*, X, H) = N_gOP_1(anti_{2'}, X, H) = NREG$.

Recently, for minimal parallelism, the following result was obtained, see [7]:

Theorem 6. $N_gOP_3(anti_2, min, H) = NRE$.

We shall improve this result by showing that only two membranes are needed:

Theorem 7. $Ps_gOP_2(anti_{2'}, min, H) = PsRE$.

Proof. We only give a sketch of the proof, because the basic ideas are the same as in the usual proofs showing computational completeness for antiport P systems. Now let $M = (n, B, P, p_0, p_h)$ be a register machine generating an output vector of dimension k ($\leq n$); then we construct the P system

$$
\begin{aligned}
\Pi &= (V, T, V, \mu, p_0, ZX, R_1, R_2, 2)\,,\\
V &= \{p, p', p'', p''', \tilde{p}, \tilde{p}', \tilde{p}'', \bar{p}, \bar{p}', \bar{p}'' \mid p \in B\}\\
&\quad \cup \{X, Y, Z, Z'\} \cup \{A_i \mid 1 \le i \le n\}\,,\\
T &= \{A_i \mid 1 \le i \le k\}\,,\\
\mu &= [_1\, [_2\ \ ]_2\,]_1,\\
R_1 &= R_{1,A} \cup R_{1,S} \cup R_{1,F},\\
R_{1,A} &= \{(p, out; A_r q, in)\,, (p, out; A_r s, in) \mid p : (A\,(r)\,, q, s) \in P\}\,,\\
R_{1,S} &= \{(p, out; p'p'', in), (p''A_r, out; p''', in), (p''X, out; \bar{p}, in),\\
&\quad (p'''X, out; \tilde{p}, in), (\bar{p}, out; \bar{p}'X, in), (\bar{p}', out; \bar{p}''Y, in),\\
&\quad (\bar{p}'', out; s, in), (\tilde{p}, out; \tilde{p}'X, in), (\tilde{p}', out; \tilde{p}''Y, in),\\
&\quad (\tilde{p}'', out; q, in) \mid p : (S\,(r)\,, q, s) \in P\}\,,\\
R_{1,F} &= \{(p'Y, out; Z', in) \mid p \in B \setminus \{p_h\}\} \cup \{(Z', out)\,, (ZX, out; Z', in)\}\,,\\
R_2 &= R_{2,A} \cup R_{2,S} \cup R_{2,F},\\
R_{2,A} &= \{(A_i, in) \mid 1 \le i \le k\}\,,\\
R_{2,S} &= \{(X, out; p', in) \mid p \in B \setminus \{p_h\}\} \cup \{(Z, out; XY, in)\,, (Z, in)\}\,,\\
R_{2,F} &= \{(p'Y, out; p_h, in) \mid p \in B \setminus \{p_h\}\} \cup \{(ZX, out; p_h, in)\,, (p_h, out)\}\,.
\end{aligned}
$$

An ADD instruction $p : (A\,(r)\,, q, s) \in P$ is simulated by using one of the rules $(p, out; A_r q, in)$, $(p, out; A_r s, in)$ assigned to membrane 1; in case r is an output register, the terminal symbol A_r is moved into the output region 2 by using (A_r, in) from R_2. A SUB instruction $p : (S\,(r)\,, q, s) \in P$ is simulated by using the rules from $R_{1,S}$ and $R_{2,S}$ in parallel. The final procedure in Π starts when the final label p_h appears; as the number of symbols p' equals the number of symbols Y as they have been introduced when simulating a SUB instruction, we finally eliminate pairs $p'Y$ from the system using the rules from $R_{1,F}$ and $R_{2,F}$ until finally only p_h remains in the skin membrane and the desired output is found in the second membrane region, without any additional symbols remaining there anymore.

The general result in Theorem 1 and the special result in Theorem 4 immediately yield the following one:

Corollary 1. $Ps_gOP_1\,(anti_{2'}, max, h) = DPs_aOP_1\,(anti_{2'}, max, h) = PsRE.$

With the other derivation modes and partial halting, we only get Parikh sets of matrix languages (regular sets of non-negative integers), which is an immediate consequence of Theorem 3:

Corollary 2. *For every* $X \in \{min, asyn, sequ\}$,
$Ps_gOP_*\,(anti_*, X, h) = Ps_gOP_1\,(anti_{2'}, X, h) = PsMAT^{\lambda}$ *and*
$N_gOP_*\,(anti_*, X, h) = N_gOP_1\,(anti_{2'}, X, h) = NREG.$

For symport rules, the following result is known (e.g., see [22]):

Theorem 8. $Ps_gOP_2\,(sym_2, max, H) = Ps_aOP_2\,(sym_2, max, H) = PsRE.$

Computational completeness can also be obtained with minimal parallelism and total halting, whereas as a direct consequence of Theorem 3, we only get Parikh sets of matrix languages (regular sets of non-negative integers) with partial halting:

Theorem 9. $Ps_gOP_2\left(sym_3, min, H\right) = PsRE.$

Proof. Let $M = (n, B, P, p_0, p_h)$ be a register machine generating an output vector of dimension k ($\leq n$); then we construct the P system

$$
\begin{aligned}
\Pi &= (V, T, E, \mu, p_0, w_1, w_2, R_1, R_2, 2)\,,\\
V &= \{p, p', p'', p''', \tilde{p}, \tilde{p}', \tilde{p}'', \tilde{p}''', \bar{p}, \bar{p}', \bar{p}'', \bar{p}''', \hat{p}, \hat{p}' \mid p \in B\}\\
&\quad \cup \{A_i \mid 1 \leq i \leq n\}\,,\\
T &= \{A_i \mid 1 \leq i \leq k\}\,,\\
E &= V \setminus \{p'_h\} \cup \{p', p''', \tilde{p}'', \bar{p}, \bar{p}', \bar{p}''', \hat{p}' \mid p \in B\}\\
\mu &= [_1\, [_2\,]_2\,]_1,\\
w_1 &= \{p_0, p'_h\} \cup \{p', p''', \tilde{p}'', \bar{p}, \bar{p}''' \mid p \in B\}\,,\\
w_2 &= \{\hat{p}', \bar{p}' \mid p \in B\}\,,\\
R_1 &= R_{1,A} \cup R_{1,S} \cup R_{1,F},\\
R_{1,A} &= \{(pp', out)\,, (p' A_r p'', in)\,, (p'' p''', out)\,, (p''' q, in)\,, (p''' s, in)\\
&\qquad \mid p : (A\,(r)\,, q, s) \in P, 1 \leq r \leq k\}\\
&\quad \cup \{(pp', out)\,, (p' A_r q, in)\,, (p' A_r s, in)\\
&\qquad \mid p : (A\,(r)\,, q, s) \in P, k < r \leq n\}\,,\\
R_{1,S} &= \{(pp', out)\,, (\tilde{p}\bar{p}' p', in)\,, (\tilde{p}' \tilde{p}'' A_r, out)\,,\\
&\qquad (\tilde{p}'' \tilde{p}''', in)\,, (\bar{p} p''' \tilde{p}''', out)\,, (\bar{p} p''' q, in)\,,\\
&\qquad (\tilde{p}\bar{p}' \bar{p}', out)\,, (\bar{p}' \bar{p}'', in)\,, (\bar{p}\bar{p}'' \bar{p}''', out)\,, (\bar{p}\bar{p}''' s, in)\\
&\qquad \mid p : (S\,(r)\,, q, s) \in P\}\,,\\
R_{1,F} &= \{(\hat{p}_i \hat{p}'_i \bar{p}'_i, out) \mid 1 \leq i \leq l\} \cup \left\{\left(\hat{p}'_i \hat{p}'_{i+1}, in\right) \mid 1 \leq i < l\right\}\\
&\quad \cup \{(p_h p'_h, out)\,, (p'_h \hat{p}_1, in)\}\,,\\
R_2 &= R_{1,A} \cup R_{1,S} \cup R_{2,F},\\
R_{2,A} &= \{(A_i, in) \mid 1 \leq i \leq k\}\,,\\
R_{2,S} &= \{(\tilde{p}\bar{p}, in)\,, (\tilde{p}\bar{p}', out)\,, (\bar{p}' \tilde{p}''', in)\,, (\bar{p}' \bar{p}'', in)\,,\\
&\qquad (\bar{p}\tilde{p}''', out)\,, (\bar{p}\bar{p}'', out) \mid p : (S\,(r)\,, q, s) \in P\}\,,\\
R_{2,F} &= \{(\hat{p}_i, in)\,, (\hat{p}_i, out)\,, (\hat{p}_i \hat{p}'_i \bar{p}'_i, out) \mid 1 \leq i \leq l\}\,.
\end{aligned}
$$

An ADD instruction $p : (A\,(r)\,, q, s) \in P$, $k < r \leq n$, is simulated by sending out the label of the instruction p together with p' which returns with A_r as well as the label of the next instruction q or s to be simulated. The simulation of an ADD instruction $p : (A\,(r)\,, q, s) \in P$, $1 \leq r \leq k$, takes two steps more – after sending out p, p' we return with p' and A_r as well as with p'' which is sent out together with p'''; p''' then returns with the label of the next instruction q or s to be simulated, whereas in the meantime A_r has got the chance to enter membrane 2.

In the case of a SUB instruction $p : (S\,(r)\,, q, s) \in P$, p' returns with $\tilde{p}$ and $\tilde{p}'$. Whereas $\tilde{p}$ enters membrane 2 together with $\bar{p}$, $\tilde{p}'$ gets the chance to take one copy of A_r out of region 1 using the rule $(\tilde{p}' \tilde{p}'' A_r, out)$. Depending on whether this rule had to be applied or not, the simulation proceeds until finally the label of the corresponding instruction to be simulated next is brought in together with p''' or $\bar{p}'''$, respectively. We should like to mention that all the symbols from $V \setminus E$ used during the simulation finally have returned to their original locations.

When the computation of the register machine stops, the label p_h appears; in order to clean the elementary membrane region 1 from non-terminal symbols we

have to use the rules from $R_{1,F}$ and $R_{2,F}$; the labels from B are assumed to be ordered in a sequence p_1 to p_l. The symbols $\bar{p}'_i$ are taken out one after the other using the rules $(\hat{p}_i\hat{p}'_i\bar{p}'_i, out)$; the rules $(\hat{p}_i, in)$ and $(\hat{p}_i, out)$ could be used for an unbounded number of steps, but for obtaining a terminating computation at some moment the rule $(\hat{p}_i\hat{p}'_i\bar{p}'_i, out)$ has to be used twice. The computation halts when all $\bar{p}'_i$ have been taken out.

Corollary 3. *For every* $X \in \{min, asyn, sequ\}$,
$Ps_gOP_*\,(sym_*, X, h) = PsMAT^{\lambda}$ *and* $N_gOP_*\,(sym_*, X, h) = NREG$.

3.3 Results for P Systems with Conditional Uniport Rules

Using only conditional uniport rules of type $uni_{1,1}$, we again obtain computational completeness, even with the minimally parallel derivation mode, together with total halting, whereas, as a direct consequence of Theorem 3, with partial halting we only get Parikh sets of matrix languages (regular sets of non-negative integers) with the minimally parallel derivation mode.

Theorem 10. $Ps_gOP_{13}\,(uni_{1,1}, X, H) = PsRE$, *for every* $X \in \{min, max\}$.

Proof. We only give a sketch of the proof showing how register machine instructions can be simulated, using the following membrane structure:

$$\left[_1\left[_2\left[_{2'}\right]_{2'}\right]_2\left[_3\left[_6\right]_6\left[_7\left[_{7'}\right]_{7'}\right]_7\left[_8\left[_{8'}\right]_{8'}\right]_8\right]_3\left[_4\right]_4\left[_5\left[_{5'}\left[_{5''}\right]_{5''}\right]_{5'}\right]_5\right]_1$$

Throughout the proof we will use a specific notation for the conditional uniport rules to illustrate the trajectories of the objects, in a more relaxed way also allowing symport rules of weight one (observe that we could use dummy objects as conditions). Now let i be a membrane and $\hat{\imath}$ be the surrounding membrane (for the skin membrane 1 this is the environment 0) ; then we use the following notations (a, b are arbitrary objects):

$(i, a, \hat{\imath})$	represents	$\left[_i{}^{a} \rightarrow {}^{a}\left[_i\right.\right.$,
$(\hat{\imath}, a, i)$	represents	${}^{a}\left[_i\right. \rightarrow \left[_i{}^{a}\right.\right.$,
$(i, a \rightarrow \hat{\imath}, b)$	represents	${}^{b}\left[_i{}^{a} \rightarrow {}^{ab}\left[_i\right.\right.$,
$(\hat{\imath}, a \rightarrow i, b)$	represents	${}^{a}\left[_i{}^{b} \rightarrow \left[_i{}^{ab}\right.\right.$,
$(i, a\,[b] \rightarrow \hat{\imath})$	represents	$\left[_i{}^{ab} \rightarrow {}^{a}\left[_i{}^{b}\right.\right.$,
$(\hat{\imath}, a\,[b] \rightarrow i)$	represents	${}^{ab}\left[_i\right. \rightarrow {}^{b}\left[_i{}^{a}\right.\right.$.

Moreover, we will use the shortcut notation

$$(A, B; i_1, \ldots, i_k; j_1, \ldots, j_{k-1})$$

which corresponds to the following group of rules:

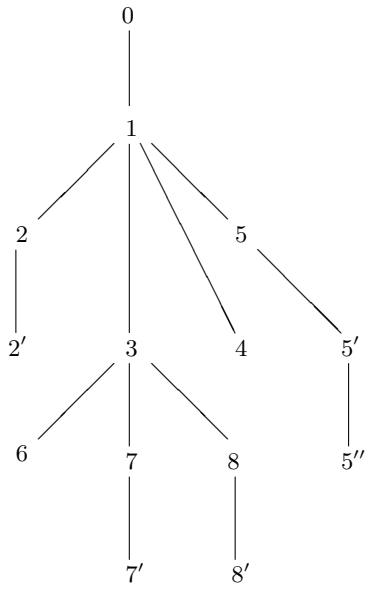

Fig. 1. Membrane structure

$$
\begin{array}{l}
(i_t, B, j_t) \\
(j_t, B, j'_t) \\
(j'_t, B, j_t) \\
(i_t, A[B], i_{t+1}) \\
(i_t, B \to i_{t+1}, A) \\
\text{for all } 1 \leq t \leq k-1
\end{array}
$$

Informally, this means that the symbols A and B travel together from membrane i_1 to membrane i_k following the path $i_1, \ldots, i_k$. The rules above permit to implement this behavior in $2k$ steps, because if something else happens, then symbol B will be trapped between membranes j_t and j'_t. Of course, we assume that B can never go out from membrane j_t and that B cannot interact with other symbols during its move along the path.

The system is constructed as follows: membrane 8 holds the current state, membrane 7 holds values of registers, membranes 4, 5 and 6 hold additional symbols. Membrane 2 and all primed membranes are used to trap symbols.

For simulating an ADD instruction, the state symbol from membrane 8 travels with an accompanying symbol (from membrane 6) to membrane 5. The accompanying symbol then brings a second accompanying symbol from the environment and both of them move the new state and a copy of the register object A_r to the corresponding membranes. After that, the first accompanying symbol returns to its original location.

The SUB instruction is simulated in an even easier way. The state symbol brings two symbols from membrane 6 to membrane 3. While it travels with one of these symbols to membrane 5, the second one tries to decrement the register. Now depending on the position of this second symbol (membrane 3 if the register is zero, or membrane 1 if the register is not zero) the corresponding new state is chosen.

We remark, that finally all involved symbols return to their original locations. Moreover, we always move groups of two symbols and if a rule not conforming to the scenario above is applied, then one of these symbols is trapped.

In more detail, for the simulation of an ADD instruction $p : (A(r), q, s)$ the following rules are used:

$I.$ $(I_p, p; 3, 1, 5; 7, 2)$
$II.$ $(T_p, I_p; 5, 1; 5')$
$III.$ $(I_p, A_r; 1, 3, 7; 2, 8)$
$IV.$ $(X_p, q; 1, 3, 8; 2, 7)$
$V.$ $(X_p, s; 1, 3, 8; 2, 7)$

1. $(8, p, 3)$
2. $(6, I_p \to 3, p)$
3. $(3, I_p, 6)$
4. $(7, I_p, 3)$
5. $(0, A_r \to 1, I_p)$
6. $(0, X_p \to 1, T_p'')$
7. $(5, T_p' \to 1, T_p)$
8. $(5, q \to 1, T_p)$
9. $(5, s \to 1, T_p)$
10. $(1, T_p', 5)$
11. $(1, T_p[q], 5)$
12. $(1, T_p[s], 5)$
13. $(4, T_p'' \to 1, T_p')$
14. $(1, T_p'', 4)$
15. $(1, T_p, 2)$
16. $(1, X_p, 2)$
17. $(2, T_p, 2')$
18. $(2, X_p, 2')$
19. $(2', T_p, 2)$
20. $(2', X_p, 2)$
21. $(8, X_p, 8')$

The simulation usually starts with p in membrane 8, with I_p being in membrane 6, and q, s in membrane 5, respectively.

Symbol p goes to membrane 3 and brings there symbol I_p from membrane 6. After that they both travel to membrane 5. There symbol I_p is moved together with symbol T_p to membrane 1. After that I_p brings one A_r from the environment and they travel together to membrane 7 and continue afterwards to membrane 3 and 6. In the meanwhile, symbol T_p in membrane 1 brings T_p' and q or s (in this order, otherwise the state symbol will be trapped in membrane 2). At the same time when q (or s) is brought in membrane 1, symbol T_p'' is brought into membrane 1 by T_p'. After that symbol T_p is sent to membrane 5 and at the same time symbol X_p is brought into membrane 1 by the symbol T_p''. Finally, symbols X_p and q (or s) go together to membrane 8. We remark that the rules 3, 10 or 14 may be used instead of some rules of the chain presented above. But in this case symbol p or symbol T_p will be trapped. We also remark that X_p and q will be ready to move to membrane 3 after I_p and A_r have arrived there.

The simulation of a SUB instruction $p : (A(r), q, s)$ is performed by the following rules:

$I.\ (M'_p, p; 1, 5; 2)$
$II.\ (M_p, A; 3, 1; 8)$
$III.\ (q, M'_p; 5, 1; 5')$
$IV.\ (s, M'_p; 5, 1; 5')$
$V.\ (M_p, q; 1, 3; 2)$

1. $(8, p, 3)$
2. $(6, M_p \to 3, p)$
3. $(6, M'_p \to 3, p)$
4. $(3, M'_p[M_p], 6)$
5. $(3, M_p \to 6, M'_p)$
6. $(7, A_r \to 3, M_p)$
7. $(3, M_p[p], 1)$
8. $(3, p \to 1, M'_p)$
9. $(1, M'_p, 3)$
10. $(1, s \to 3, M_p)$
11. $(1, A_r[M_p], 0)$
12. $(3, q[M_p], 8)$
13. $(3, s[M_p], 8)$
14. $(3, p, 7)$
15. $(7, p, 7')$
16. $(7', p, 7)$

The simulation usually starts with object p in membrane 8, whereas M_p, M'_p are found in membrane 6, q, x in membrane 5, and A_r possibly in membrane 7.

Symbol p first goes from membrane 8 to membrane 3 and after that brings there symbol M'_p. After that it moves symbol M'_p to membrane 1 and brings symbol M_p to membrane 3. Further, p moves to membrane 1 and M_p may bring a symbol A_r to membrane 3. If it succeeds, then both symbols M_p and A_r move to membrane 1 and after that symbol A_r is sent out. In the meanwhile p and M'_p move to membrane 5. From there, M'_p brings either q or s to membrane 1. Now if it brought q and the register was not zero (M_p is in membrane 1) then M_p will bring q to membrane 8. Otherwise, q will be trapped in membrane 2. If s was brought into membrane 1 by M'_p, and the value of register was zero, then this s will move to membrane 3 and further to membrane 8. It is easy to observe that the symbols M_p and M'_p return to membrane 6 by themselves. They can do this at any moment of the computation, but only after the state symbol q or s has returned to membrane 8 they can do this without provoking an infinite computation.

3.4 Results for Evolution/Communication P Systems

For evolution/communication P systems, the constructions from [2], Theorems 1 and 2, and from [3], Theorems 4.3.1 and 4.3.2, already show the computational completeness, using two membranes, for the minimally parallel setup (when working in the maximally parallel way, the system never applies simultaneously more than one rule from the same set of rules assigned to a membrane):

Corollary 4. *For* $X \in \{min, max\}$,
$Ps_gOP_2((ncoo, anti_1, sym_1), X, H) = PsRE,\ X \in \{min, max\}$.

We can extend these results by showing that *deterministic* evolution-communication P systems with non-cooperative evolution rules and communication rules of weight one (also see [1]) are computationally complete, using three membranes.

Theorem 11. *For* $X \in \{min, max\}$,
$DPs_aOP_3((ncoo, sym_1, anti_1), X, H) = PsRE$.

Proof. Consider a deterministic register machine $M = (n, B, P, p_0, p_h)$. Let us denote the set of labels of SUB instructions by B_-.

$$
\begin{aligned}
\Pi &= \left(V, T, V, [_1\, [_2\,]_2 [_3\,]_3\,]_1, p_0, \lambda, \lambda, R_1, R_2, R_3, 1\right), \\
V &= B \cup \{l_j \mid l \in B_-, 0 \le j \le 7\} \cup \{q\} \\
&\quad \cup \{A_i \mid 1 \le i \le n\} \cup \{i_j \mid 1 \le i \le n,\ 1 \le j \le 3\}, \\
T &= \{A_i \mid 1 \le i \le k\}, \\
R_1 &= \{l \to A_i l' \mid (l : A(i), l') \in P\} \cup \{l_j \to l_{j+1} \mid l \in B_-,\ j \in \{0,1,2,5,6\}\} \\
&\quad \cup \{l \to l_0 i_1,\ l_4 \to l_5 q,\ l_7 \to l'' \mid l : (S(i), l', l'') \in P\}, \\
R_2 &= \{i_1 \to i_2,\ i_3 \to \lambda \mid 1 \le i \le n\} \cup \{l_3 \to l_4 \mid l \in B_-\}, \\
&\quad \cup \{(i_1, in),\ (i_2, out; A_i, in),\ (i_3, in) \mid 1 \le i \le n\} \\
&\quad \cup \{(i_2, out; l_3, in),\ (l_4, out) \mid l : (S(i), l', l'') \in P\}, \\
R_3 &= \{i_2 \to i_3 \mid 1 \le i \le n\} \cup \{q \to \lambda\} \cup \{l_3 \to l' \mid l \in B_-\} \\
&\quad \cup \{(i_2, in),\ (i_3, out; q, in) \mid 1 \le i \le n\} \\
&\quad \cup \{(i_3, out; l_3, in),\ (l', out) \mid l : (S(i), l', l'') \in P\}.
\end{aligned}
$$

An ADD instruction $l : (A(i), l') \in P$ is implemented by the single non-cooperative evolution rule $l \to A_i l'$.

The simulation of a SUB instruction $l : (S(i), l', l'') \in P$ is described in a depictive way in the following tables. Decrementing register i works as follows:

step	0	1	2	3	4
region 2			i_1	i_2	a_i
rules 2		(i_1, in)	$i_1 \to i_2$	$(i_2, out; A_i, in)$	
region 1	lA_i	$l_0 i_1 A_i$	$l_1 A_i$	$l_2 A_i$	$l_3 i_2$
rules 1	$l \to l_0 i_1$	$l_0 \to l_1$	$l_1 \to l_2$	$l_2 \to l_3$	
region 3					
rules 3					(i_2, in)

step	5	6	7	8	
region 2				i_3	
rules 2			(i_3, in)	$i_3 \to \lambda$	
region 1	l_3	l_3	i_3		l'
rules 1					
region 3	i_2	i_3	l_3	l'	
rules 3	$i_2 \to i_3$	$(i_3, out; l_3, in)$	$l_3 \to l'$	(l', out)	

If register i is empty, i.e., if there is no object A_i, the simulation works as follows:

step	0	1	2	3	4	
region 2			i_1	i_2	i_2	
rules 2		(i_1, in)	$i_1 \to i_2$		$(i_2, out; l_3, in)$	
region 1	l	$l_0 i_1$	l_1	l_2	l_3	
rules 1	$l \to l_0 i_1$	$l_0 \to l_1$	$l_1 \to l_2$	$l_2 \to l_3$		
region 3						
rules 3						

step	5	6	7	8	9	10
region 2	l_3	l_4				i_3
rules 2	$l_3 \to l_4$	(l_4, out)			(i_3, in)	$i_3 \to \lambda$
region 1	i_2		l_4	$l_5 q$	$l_6 i_3$	l_7
rules 1			$l_4 \to l_5 q$	$l_5 \to l_6$	$l_6 \to l_7$	$l_7 \to l''$
region 3		i_2	i_3	i_3	q	
rules 3	(i_2, in)	$i_2 \to i_3$		$(i_3, out; q, in)$	$q \to \lambda$	

The main idea of the construction is similar to that in [1]: if the object A_i is present in region 2, it is exchanged with object i_2 while object l_2 changes to l_3; otherwise object l_3 is exchanged with i_2 in the next step. The trajectory of object i_2 is the same in both cases: it enters membrane 3, is renamed, exits and enters membrane 2, where it is erased. The behavior of object l_3 indirectly depends on the presence of A_i via i_2 or i_3: in the decrement case it enters membrane 3, is renamed to l' and returns to region 1, while in the zero-case it enters membrane 2, is renamed to l_4, exits, and produces two objects; one of them helps i_3, while the other one produces l'', thus finishing the simulation.

4 Conclusion

In this paper, we have investigated a new variant of halting – we call it *partial halting* – in membrane systems where all membranes are required to allow for the application of a rule at the same time in order to keep a computation alive. Obviously, for systems with only one membrane this way of halting is equivalent with the original one where a system halts if and only if no rule is applicable anymore in the whole system – we also call this *total halting*. Besides this general result, we also have shown that P systems working in the minimally parallel mode, the asynchronous or the sequential derivation mode and with partial halting can only generate Parikh sets of matrix languages/regular sets, the same what we obtain with the sequential and the asynchronous derivation mode and total halting.

Comparing the results for total and partial halting for the minimally parallel derivation mode elaborated above, we realize that for any of the specific restricted variants α of P systems with permitting contexts we have

$$Ps_gOP_*(\alpha, min, h) \subseteq PsMAT^{\lambda} \subsetneqq PsRE = Ps_gOP_*(\alpha, min, H) \text{ and}$$
$$N_gOP_*(\alpha, min, h) = NREG \subsetneqq NRE = N_gOP_*(\alpha, min, H),$$

i.e., in the case of the minimally parallel derivation mode the halting condition – *total* in contrast to *partial* halting – makes the difference. Intuitively speaking, the requirement for a computation to continue only if for every membrane a rule is applicable, together with the minimally parallel derivation mode means that we do not have the possibility of appearance checking and therefore cannot simulate the zero test for register machines, hence, we cannot obtain computational completeness.

In the future, the new variant of partial halting should also be investigated for other variants of P systems working in the different derivation modes, with multisets of objects, but also with strings, arrays, etc.

Acknowledgements.

Artiom Alhazov gratefully acknowledges support by the Academy of Finland, project 203667; he also acknowledges the project 06.411.03.04P from the Supreme Council for Science and Technological Development of the Academy of Sciences of Moldova. The work of Marion Oswald was supported by FWF-project T225-N04. 2006.

References

1. A. Alhazov: On determinism of evolution-communication P systems, *Journal of Universal Computer Science* **10**, 5, 2004, 502–508.
2. A. Alhazov, Number of protons/bi-stable catalysts and membranes in P systems. Time-freeness. In: [15], 79–95.
3. A. Alhazov: Communication in Membrane Systems with Symbol Objects, Ph.D. Thesis, Tarragona, Spain, 2006.
4. F. Bernardini, V. Manca: P systems with boundary rules. In: [21], 107–118.
5. F. Bernardini, F. J. Romero-Campero, M. Gheorghe, M.J. Pérez-Jiménez, M. Margenstern, S. Verlan, N. Krasnogor: On P systems with bounded parallelism. In: G. Ciobanu, Gh. Păun (Eds.): Pre-Proc. of First International Workshop on Theory and Application of P Systems, Timisoara, Romania, September 26–27, 2005, 31–36.
6. M. Cavaliere: Evolution-communication P systems. In: [21], 134–145.
7. G. Ciobanu, Linqiang Pan, Gh. Păun, M.J. Pérez-Jiménez: P systems with minimal parallelism, *accepted for TCS*.
8. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
9. J. Dassow, Gh. Păun: On the power of membrane computing, *Journal of Universal Computer Science* **5** (2) (1999), 33–49.
10. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**, 1–3 (2002), 81–102.
11. R. Freund, M. Oswald: P Systems with activated/prohibited membrane channels. In: [21], 261–268.
12. R. Freund, M. Oswald: P systems with conditional communication rules assigned to membranes, *Journal of Automata, Languages and Combinatorics* **9**, 4 (2004), 387–397.

13. R. Freund, M. Oswald: P systems with partial halting, submitted, 2007.
14. R. Freund, Gh. Păun: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *Theoretical Computer Science* **312** (2004), 143–188.
15. R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. 6th International Workshop WMC 2005, Vienna, Austria, Lecture Notes in Computer Science **3850**, Springer-Verlag, 2006.
16. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. *Theoretical Computer Science* **330** (2005), 101–116.
17. M.L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
18. A. Păun, Gh. Păun: The power of communication: P systems with symport/ antiport, *New Generation Computing* **20**, 3 (2002), 295–306.
19. Gh. Păun: Computing with membranes, *J. of Computer and System Sciences* **61**, 1 (2000), 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`).
20. Gh. Păun: *Computing with Membranes: An Introduction,* Springer-Verlag, Berlin, 2002.
21. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing. International Workshop WMC 2002, Curteă de Argeş, Romania, Revised Papers. Lecture Notes in Computer Science **2597**, Springer-Verlag, Berlin (2003).
22. Y. Rogozhin, A. Alhazov, R. Freund: Computational power of symport/antiport: history, advances, and open problems. In: [15], 1–30.
23. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages* (3 volumes), Springer-Verlag, Berlin, 1997.
24. S. Verlan, F. Bernardini, M. Gheorghe, M. Margenstern: On communication in tissue P systems: conditional uniport. Pre-proceedings of Membrane Computing. International Workshop, WMC7, Leiden, The Netherlands, 2006, 507–521
25. The P Systems Web Page: `http://psystems.disco.unimib.it.`